(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平10－111802

(43)公開日　平成10年(1998)４月28日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/44	５３０	Ｇ０６Ｆ	9/44	５３０Ｐ
					５３０Ｍ
	9/06	５３０		9/06	５３０Ｗ

審査請求　未請求　請求項の数14　ＯＬ　（全 22 頁）

(21)出願番号　特願平9－178588

(22)出願日　平成９年(1997)７月３日

(31)優先権主張番号　０８／６７４８２８

(32)優先日　1996年７月３日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595034134
サン・マイクロシステムズ・インコーポレイテッド
Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，　Ｉｎｃ．
アメリカ合衆国　カリフォルニア州　94303　パロ　アルト　サン　アントニオ　ロード　901

(72)発明者　ブラッド　ジー．　フォウロウ
アメリカ合衆国，　カリフォルニア州，　レッドウッド　シティ，　ヒルサイド　ロード　546

(74)代理人　弁理士　長谷川　芳樹　（外５名）

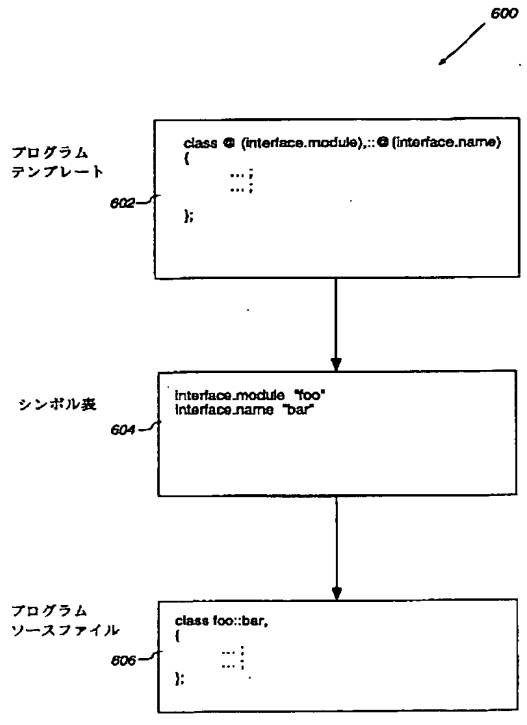
最終頁に続く

(54)【発明の名称】　分散オブジェクトシステムの応用分野におけるコード生成器

(57)【要約】

【課題】　分散オブジェクトシステムにおいて、ネットワーク上に存在するオブジェクトを利用して、アプリケーションを組み立てて行く場合に、オブジェクトの情報を定型化し、それを利用することによって、自動的にコードを生成し、アプリケーション開発の効率を高める。

【解決手段】　ネットワーク上の言語に依存しないオブジェクトを自動的に組み立てて分散オブジェクトシステム上で用いられるネットワーク・アプリケーションを作成する技術は、シンボル表とプログラム・テンプレートを用いる。　ネットワーク・アプリケーションのスキーマ表現は視覚的なアプリケーション構築器によって形成される。スキーマ表現は以前に定義された分散オブジェクトを表現したものの間に接続を定義する。　これらの接続は、　コンポーネントと名づけられた分散オブジェクト表現であるパーツ、　プラグ、　ソケットの間で形成される。



【特許請求の範囲】

【請求項１】　ネットワーク上にあり（networked）言語に独立なオブジェクト（indepentent object）を自動的に組み立てて、分散オブジェクト・システムに用いるネットワーク・アプリケーションを生成する計算機にインプリメントされる（computer-implemented）方法であって、

前記ネットワーク・アプリケーションのスキーマ表現（schematic representation）を受けるステップであって、前記スキーマ表現は前記分散オブジェクト表現の間の（among）複数のリンクを定義し、

前記ネットワーク・アプリケーションの前記スキーマ表現をシンボル表中へロードすると共に、前記スキーマ表現の部分（portion）を前記シンボル表にある複数のエントリ（entry）として格納する（store）ロードステップと、

少なくとも一つ生成されるべきプログラム・ソース・ファイルと、プログラム・ソース・ファイルの発生に使用するための対応する少なくとも一つのプログラム・テンプレートと、を決定するステップであって、前記プログラム・テンプレートは前記シンボル表にある複数のエントリ（entry）へのレファランス（reference）を含み、前記シンボル表にある複数のエントリを対応する少なくとも一つの前記プログラム・テンプレートと組み合わせ、それによって、コンパイルされ前記ネットワーク・アプリケーションの一部を形成するために適する、少なくとも一つの前記プログラム・ソース・ファイルを作るステップと、を備える方法。

【請求項２】　前記ネットワーク・アプリケーションの前記スキーマ表現は、視覚的アプリケーションビルダ（builder）内に形成され、クライアント・オブジェクトおよびサーバ・オブジェクトの一方を示すように配置される、請求項１に記載の方法。

【請求項３】　分散オブジェクトの表現の間の前記複数のリンクは、パーツ（parts）、プラグ（plugs）およびソケット（sockets）を含む要素（element）の組み合わせによって形成される、請求項１または請求項２に記載の方法。

【請求項４】　スキーマ表現をロードするロードステップは、前記スキーマ表現のパーツ、プラグ、ソケットをロードすることを含む、請求項１から請求項３のいずれかに記載の方法。

【請求項５】　前記シンボル表にある前記複数のエントリの各一つは対応する値にマップする（map）識別子（identifier）を含み、且つ前記複数のエントリを組み合わる前記ステップは、前記プログラム・テンプレートにある前記識別子への前記レファランスをシンボル表からの対応する値に置換するように作用する、請求項１から請求項４のいずれかに記載の方法。

【請求項６】　少なくとも一つの前記ソースファイルがコンパイルされ、ネットワーク・アプリケーションの一部となることを要求するステップを含む、請求項１から請求項５のいずれかに記載の方法。

【請求項７】　ネットワーク上にあり言語に独立なオブジェクトを自動的に組み立てて、分散オブジェクト・システムに用いるネットワーク・アプリケーションを生成する計算機にインプリメントされる方法であって、

前記ネットワーク・アプリケーションの前記スキーマ表現を視覚的アプリケーションビルダを用いて生成するステップであって、前記スキーマ表現はクライアント・オブジェクトおよびサーバ・オブジェクトの一方を表示するように配置され、スキーマ表現は分散オブジェクトの表現の間の複数のリンクを定義し、

前記ネットワーク・アプリケーションの前記スキーマ表現をシンボル表にロードし、前記スキーマ表現の部分を前記シンボル表にある複数のエントリとして格納するステップと、

生成されるべき少なくとも一つプログラム・ソース・ファイルと、前記プログラム・ソース・ファイルを作るための対応する少なくとも一つのプログラム・テンプレートとを決定するステップであって、前記プログラム・テンプレート前記前記シンボル表にある複数のエントリのレファランスを含み、

前記シンボル表にある複数のエントリを対応する少なくとも一つのプログラム・テンプレートと組み合わせ、それによって、コンパイルされネットワーク・アプリケーションの一部を形成するために適する、少なくとも一つの前記ソース・ファイルを形成するステップと、を備える方法。

【請求項８】　分散オブジェクトの表現の間の複数の前記リンクは、パーツ、プラグ、およびソケットを含む要素の組み合わせによって形成される、請求項７に記載の方法。

【請求項９】　前記スキーマ表現をロードするステップは、前記スキーマ表現の要素をロードすることを含む、請求項８に記載の方法。

【請求項１０】　前記シンボル表にある複数の前記エントリの各一つは対応する値にマップする識別子を含み、且つ複数のエントリを組み合わせるステップは、前記プログラム・テンプレートにある前記識別子への前記レファランスを前記シンボル表にある対応する値に置換するように作用する、請求項７から請求項９のいずれかに記載の方法。

【請求項１１】　少なくとも一つの前記ソースファイルがコンパイルされ、前記ネットワーク・アプリケーションの一部となることを要求するステップを更に含む、請求項７から請求項１０のいずれかに記載の方法。

【請求項１２】　ネットワーク・アプリケーションのスキーマ表現をシンボル表にロードし、それにより、ネットワーク上にあり言語に独立なオブジェクトを自動的に

組み立てて、分散オブジェクト・システムに用いるネットワーク・アプリケーションを生成することを補助する計算機にインプリメントされる方法であって、前記シンボル表は複数の識別子を対応する値にマップするよう配置され、前記スキーマ表現は値を有する識別子に関連づけられる要素を含み、

該スキーマ表現からトップレベル・シンボルをシンボル表へロードし、前記トップレベル・シンボルに関連づけられた識別子を対応する値へマップするステップと、

該スキーマ表現から接続をシンボル表へロードし、前記接続に関連づけられた識別子を対応する値へマップするステップと、を備える方法。

【請求項１３】　スキーマ表現の構築に用いられるファイルを指示すると共に生成されるべきファイルを指示するファイル記述子（file descriptor）をロードするステップと、

該スキーマ表現からプラグを該シンボル表にロードし、前記プラグに関連づけられる識別子を対応する値にマップするステップと、

スキーマ表現からソケットをシンボル表にロードし、前記ソケットに関連づけられる識別子を対応する値にマップするステップと、を更に備える請求項１２に記載の方法。

【請求項１４】　ネットワーク上にあり言語に独立なオブジェクトを自動的に組み立てて、分散オブジェクト・システムに用いるネットワーク・アプリケーションを生成するために使用する計算機装置（computer apparatus）であって、

処理ユニットと、

前記処理ユニットに結合する入力/出力デバイス（device）と前記処理ユニットと通信する記憶デバイスと、

前記ネットワーク・アプリケーションの、分散オブジェクトの表現に間の複数のリンクを定義するスキーマ表現を受ける手段と、

前記ネットワーク・アプリケーションの前記スキーマ表現をシンボル表にロードすると共に、前記スキーマ表現の一部を前記シンボル表にある複数のエントリとして格納する手段と、

生成されるべき少なくとも一つのファイルと、前記ソースファイルを生成するために使用する対応する少なくとも一つのプログラム・テンプレートとを決定する手段であって、前記プログラム・テンプレートは前記シンボル表にある複数のエントリへのレファランスを含み、

前記シンボル表にある複数のエントリを対応する少なくとも一つのプログラム・テンプレートを少なくとも一つ組み合わせ、それによって、コンパイルされネットワーク・アプリケーションの一部を形成するために適する、少なくとも一つの前記ソースファイルを作成する手段と、を備える計算機装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】この発明は分散計算機システム、クライアント-サーバ・システム、およびオブジェクト指向プログラミングに関わるものである。具体的には、アプリケーションプログラムを分散オブジェクトシステム上に作成し、インストールする技術に関する発明である。

【０００２】

【従来の技術】オブジェクト指向に関する方法論は過去数年に渡ってますます注目を集めてきた。これは従来的なプログラミングの方法によるソフトウェア開発では時間と予算がかかり過ぎることによる。オブジェクト指向のプログラミング方法論は手続きよりもむしろデータ操作にフォーカスを置く。したがって、プログラマーにとっては実世界の問題に対してより直観的なアプローチが可能になる。加えて、オブジェクトとはそれに関係するデータと手続きをカプセル化したものであり、このオブジェクトのインタフェースにあるデータと手続きだけをアクセス可能にし、それ以外の他のプログラムの部分を切り分けておくことができる。したがって、あるオブジェクトのデータと手続き部分の変更は他の部分に影響しない。このやり方は、あるオブジェクトの仕様変更が他のオブジェクトを干渉しない点で、従来的なプログラミング方法論に比べてコードのメンテナンスをより容易にするものである。さらには、オブジェクトにはモジュールという考え方が内在しているため、あるオブジェクトを他のプログラムの中で転用することができる。したがって、プログラマーは「試用され、機能する」オブジェクトをライブラリとして貯め込み、何度も他のアプリケーションで使うことができる。信頼性のあるコードを繰り返し使うことは、ソフトウェア全体の信頼性を高め、開発時間を短縮することになる。

【０００３】

【発明が解決しようとする課題】分散オブジェクトシステムにおいてオブジェクト指向プログラミングを用いる利点は多いが、このことを実装（インプリメンテーション、implementation）するとなると大きな問題が残る。一般に、プログラミングの過程でソフトウェアの再利用することは、オブジェクト指向の世界においても難しいことである。特に、プログラマーは自分の理解があまり及ばないようなコードを用いるのを嫌がるものである。分散オブジェクトシステムではこの問題は倍化する。あるコードの開発者のコメントやインストラクションは新しいアプリケーションの開発者の耳には届きにくくなるからである。したがって、分散システムからは多くのコードが利用可能でありその恩恵を被ることが本来できるはずであるのに、実際は既存のコードを書き直すことを余儀なくされることになる。

【０００４】分散オブジェクトモデルにおいて現在のコードを共有化する技術は不十分である。現在のコード共

5

有化とは、ライブラリの使用、一般に公開されているディレクトリのヘッダ・ファイルの使用、広く流布した文書（電子的にも紙の形でも）の使用などが含まれる。これらの方法は、しかしながら、分散オブジェクト環境には充分なじまないものである。分散オブジェクト環境では、再利用される単位がファイル・ベースではなく、インタフェース・ベースなのである。さらにまた、 分散環境における再利用の方法においては、プログラマーも開発されるコードも「共同体」意識をさらに高める努力をする必要がある。このように、分散オブジェクトシステムにおけるオブジェクトの再利用をより活発化したいならば、ユーザから見たオブジェクトの識別、目的、利用法などを容易にするべきである。

【０００５】分散オブジェクト環境におけるプログラミングのもう一つの挑戦課題は、「反復使用語句（boiler plate）」型のコンピュータコードをより多く供給することである。これにより、分散オブジェクトシステムの中で動くよう開発されたオブジェクトやアプリケーションは、そのシステムの中でより確実に動作するようになる。特に、C++ オブジェクトのような通常のオブジェクトが分散オブジェクトとして機能できるような基礎的なコンピュータ・コード体系が供給される必要がある。しかしながら、既存のオブジェクトを最大限再利用したいと願うプログラマーは、 同様な、しかし多少異なった挑戦課題に直面している。プログラマーが開発中のアプリケーションの中のオブジェクトを使いたいと思ったときは、そのオブジェクトの開発者など基礎的な情報を明記し初期化情報なども合わせて持たねばならない。さらに、いろいろなメイクファイル （makefile; システムのその環境でのインストール手続きファイル）やヘッダファイル、ライブラリの依存関係などが、他のコードの中に送られる前に全部付加されていなければならない。さらにはプログラマー自身によっていろいろな例外処理機構、デバッグやトレースの情報なども付加されなければならない。繰り返すと、プログラミング技法に習熟している人たちにはよく知られていることであるが、そのようなルーチンを実装（インプリメンテーション）することは労多く、繰り返しの多い、誤りを犯しがちな作業である。適切にコード化されたオブジェクトのアプリケーションの開発は極度に時間を喰う大変な仕事である。それゆえに、このような「ハウスキーピング（housekeeping）、つまりプログラムを適切に実行させるためにしなければならないシステムタスクを行う」ようなコードの作成は、自動化されることが望ましいのである。

【０００６】不幸なことに、現存する多くのオブジェクトは分散オペレーティングシステム上に置いて機能するようには書かれていない。分散システムのためのオブジェクトの実装とは、現存するオブジェクト・ソフトウェアの再編成を要求するものとなる。このことは、既存のプログラミング・オブジェクトが分散オブジェクトシステム上で容易に利用可能でないと同様、オブジェクト・プログラミング方法論の魅力を半減させるものである。

6

【０００７】分散オブジェクトシステムの上で分散オブジェクトを実装する上では、さらにオブジェクトがインストールされた時点でその存在をアナウンスする必要がある。すべてのクライアントの要求は、オブジェクト要求ブローカー （object request broker:ORB）を通じて処理される。したがって、ORBは常にその時点で現存するオブジェクトを把握している必要がある。さらには、分散オブジェクトシステムにおいては、開発途中にあるオブジェクトと分散システム上のユーザに既に利用可能になったオブジェクトを明示して区別しておくことが望ましい。このような公開情報を供給することは、それでなくても開発の重荷と分散オブジェクトシステムの利点を利用するための「ハウスキーピング」労働に追われるプログラマーにとって、新たな困難を課すことになる。

【０００８】プログラマーにとってもユーザにとっても、分散オブジェクトを作りインストールするのに相対的に透明性のあるやり方が望ましい。ここで透明性とは、異なった計算機上、異なったプログラミング言語で開発されたオブジェクトでも、さらなるコードの書き換えなど不当な労働をユーザに課すことなく、分散システム上で利用可能にできるという意味である。このように、分散オブジェクトの形成とインストールに関しては、プログラマーやユーザが分散オブジェクトシステムの細かいことまで知らずともできるようにすることが望ましい。さらには、分散されることを念頭に置かずに開発されたオブジェクトもプログラマーやユーザから見て相対的に透明に分散オブジェクトに作り替えられればもっと喜ばしい。

【０００９】

【発明を解決するための手段】本発明の実施例は、ネットワーク上の言語に依存しないオブジェクトを自動的に組み立てネットワーク・アプリケーションとする方法に関するものである。このネットワーク・アプリケーションは分散オブジェクトシステムで使われるものとする。このネットワーク・アプリケーションのモデル,あるいはスキーマ表現は、コンポジション構築器の中で形成される。このコンポジション構築器はスキーマ表現を作成するのに適した開発ツールとなりうる。一度スキーマ表現が用意されると、ネットワーク・アプリケーションのスキーマ表現を受け取るーステップが実行される。スキーマ表現はそれ以前に定義された数多く分散オブジェクトのリンクを定義している。次に、ネットワーク・アプリケーションのスキーマ表現はシンボル表上にロードされ、スキーマ表現の部分はシンボル表のエントリとして貯えられる。次のステップでは、生成されるべきソースプログラムのファイルが少なくとも一つ決定し、そのソースファイルを生成するのに対応するプログラムのテンプレートが決まる。このテンプレートはシンボル表の複

数のエントリへのレファランスを含んでいる。次のステップでは、シンボル表の多数のエントリが、少なくとも一つのプログラムテンプレートに組み合わされ、対応するプログラム・ソースを生成することになる。このプログラムのソースファイルはコンパイルするとネットワーク・アプリケーションの一部になるようにできている。

【００１０】ある実施例では、スキーマ（schematic）表現は視覚的なアプリケーション作成器によって形成され、クライアント・オブジェクトかサーバ・オブジェクトに整形される。また、分散オブジェクトの複数リンクはパーツ、プラグ、ソケットといったコンポーネントの組み合わせによって形成される。ここでロードを行なうモジュールでは、スキーマ表現のパーツ、プラグ、ソケットをシンボル表にロードする過程を含む。

【００１１】シンボル表の実施例では、シンボル表は対応する値にマップする識別子を含む。多数のエントリを結びつける操作は、プログラム・テンプレートの中の識別子へのレファランスをシンボル表からの適当な値に置換することで行なわれる。

【００１２】他の実施例では、ネットワーク・アプリケーションのスキーマ表現を視覚的なアプリケーション生成器の中に生成するステップと、プログラム・ソース・ファイルをネットワーク・アプリケーションにコンパイルし実行可能形式にリンクするようなオブジェクト開発機能を要求するステップを含む。

【００１３】本発明の一つの実施例は、シンボル表の情報をロードする手法に関するものである。この実施例はネットワーク・アプリケーションのスキーマ表現をロードする計算機による方法である。これにより、ネットワーク上の言語に依存しないオブジェクトを分散オブジェクト・システムの中で使用するネットワーク・アプリケーションに仕立てあげるのを自動化するのを補助する。シンボル表は多数の識別子を対応する多数の値にマップするよう変換される。スキーマ表現は、値を持つ識別子に対応する要素を含んでいる。この方法はどの順序で実行されてもよい、いろいろなステップから成る。最初のステップでは、スキーマ表現のトップレベルにあるシンボルはシンボル表にロードされ、トップレベル・シンボルに関連した識別子は対応する値を持つようになる。二番目のステップでは、スキーマ表現のパーツがシンボル表上にロードされ、そのパーツに関連した識別子は対応する値にマップされる。三番目のステップでは、スキーマ表現の接続関係はシンボル表上にロードされ、接続に関連した識別子は対応する値にマップされる。他の実施例では、ファイル記述子、プラグ、ソケットを同様なやり方でロードするステップを含む。

【００１４】

【発明の実施の形態】

1. 物理的実施例と分散オブジェクトシステムの背景

本発明は分散オブジェクトシステムに向けてのものであり、添付した図のようなある望ましい形での実施例に関して詳述を行なう。本発明は、CORBA のもとで定義されたものかあるいは適切な仕様のものであれば、どの分散オブジェクトシステムであれば実行可能となる。しかしながら、図で示したように、本発明の実施例に関しては、オブジェクト・管理・グループ（Object Management Group:OMG） 改訂２．０版（1995年7月）によるCORBAの仕様で実装されたオブジェクト要求ブローカ（Object Request Broker:ORB） を主に念頭に置いて行なう。図１は本発明の実施例を実装する上で適当な代表的な分散オブジェクトシステムの全体像を示している。

【００１５】分散オブジェクトシステム10は、図１に記号で示したように、典型的にはORB（Object Request Broker）11を含んでいる。ORB 11はクライアントからの呼びかけをサーバ（遠隔オブジェクト） に伝えその答を返すのに必要なすべての情報、すなわち、位置、送信メカニズム、設備を供給する。クライアントとサーバは同じプロセスに同居している場合もあれば、同じマシンの違うプロセス、異なるマシンの場合もある。議論の都合上、クライアント20は分散オブジェクトに何か操作を起こすコードであるとし、それ自身が分散オブジェクトやプロセスの形態をとっているかどうかは問わないことにする。分散オブジェクトの表現にはいろいろな方法がありうる。この例では分散オブジェクトはアプリケーション開発者によって用意された C++ のオブジェクトであってもよい。あるいは、分散オブジェクトは視覚的アプリケーション構築器（視覚的アプリケーションビルダ）15で実装されたものかも知れない。この視覚的アプリケーション構築器は、新オブジェクトを実装するにあたり、開発者に画面からカタログでオブジェクトの型を選別し、他からの必要に応じてそのオブジェクトのもつサービス機能 （属性、引数、結果など）を接合するものである。

【００１６】オブジェクト開発器（オブジェクト開発ファシリティ、object development facility）16は分散オブジェクトを単に生成し、インストールするものである。これは開発したオブジェクトを分散オブジェクトのコードに包み込みカプセル化するものである。このように、オブジェクト開発器16は開発したオブジェクトをORB オブジェクト14に変換する。この例では、ORBオブジェクト14は図の中の位置からサーバとして示されている。開発者はORBオブジェクトとのインタフェースを定義するためにインタフェース定義言語を用い、開発用オブジェクトのふるまいを決めて実装し、そののちにオブジェクト開発器16を用いて ORBオブジェクト14を実装する。このORBオブジェクト（サーバント・オブジェクト）のインスタンスは実行時に生成されるが、その際には実装されたORBオブジェクトが利用される。オブジェクト開発器はある時点でクライアントの役を受け持つオブジェクトを生成することも可能である。

【００１７】クライアント20はスタブ(stub)21、下請け層（サブコントラクト、subcontract）36、ときによりフィルタ40、トランスポート層38を通じてサーバントと通信する。スタブ21はサロゲート（surrogate）22、メソッド・テーブル24、スタブ関数25からなる。クライアント20は最初クライアントにとってサーバントに見えるサロゲート22と通信する。あるいは、クライアント20はサロゲート22、メソッド・テーブル 24、スタブ関数25を介する代わりに、動的起動（動的呼び出し）インタフェース(Dynamic Invocation Interface:DII) 26を通じて直接通信する場合もある。DII 26はクライアントが動的に要求を作るのを可能にする。

【００１８】下請け層36は特定の下請け名で指定されたいろいろなサービス（あるいは属性やオブジェクト・メカニズム）を実装する下請けを利用するためにオブジェクトから来る要求を受け付ける機能を有している。下請けは、ある個別のオブジェクトにより利用されるかも知れない分散オブジェクトシステムによって供給されるサービスの質を決定する。例えば、下請けはある特定オブジェクトによって使われる安全性の特徴を同定したりする。それぞれの下請けは実行時に動的にサーバント・オブジェクトに結びつけられる。フィルタ40が使われると、いろいろなタスク、例えば圧縮、暗号化、トレースやデバッグなどを実行し、オブジェクトとのやりとりに使われる。トランスポート層 38は、一般にはクライアントとは別のプロセスにあるサーバントへ情報を転送したり物理的に流したりする。

【００１９】標準実装スイート（標準インプリメンテーションスイート、standard implementation suite）28（オブジェクト・アダプタ）は、例えばオブジェクト・キー・マネジメントのように、ORBオブジェクト14と相互作用する下請けの一群を表わす。ここで、下請けは複数の実装スーツに属することもある。スケルトンは、静的なスケルトン32と動的なスケルトン30の両方の形態をとることがあり、要求をサーバント・オブジェクトに合う形に変換するのに用いられる。したがって、スケルトン30と32は適当なサーバント・オブジェクトと呼ばれる。静的スケルトン32はインタフェースに特定したオブジェクトの実装14を呼び出すのに用いられ、動的スケルトン30はインタフェースを特定できるオブジェクトがないときに用いられる。ORBインタフェース34は、直接ORBのところに行くインタフェースであり、これはどのORBにも共通である。すなわち、 オブジェクトのインタフェースやオブジェクト・アダプタに依存しない。ORBデーモン46はオブジェクト・サーバがクライアントから呼び出された時、それが稼働中であることを確認する機能を持つ。

【００２０】信頼性プロトコル（安全プロトコル）42はORB のインターネット間のプロトコルを確保するものであり、トランスポート層38を通じて情報が安全に移送できるようにするためのものである。これは統合性保護、機密性などを意味する。インターネットのORB 間プロトコルは異なるマシン上のプロセス間での通信を行なうためのものであるが、ときには同じマシン上のプロセス間でも用いられることがある。セキュリティ・ サーバ54は安全性を管理するサーバで、異なるコンピュータ上のプロセス間で用いられるサービスの安全性を確保するためのものである。

【００２１】Typecode/Any モジュール44は「Typecode」および「Any」オブジェクトを実装するためのものである。Typecodeはインタフェース定義言語（IDL）のデータタイプを記述し、そこではタイプ記述がクライアントとサーバ間で転送されることを許す。IDLデータタイプのインスタンスは「Any」オブジェクトによってカプセル化される。Anyオブジェクトはカプセルの中のデータのタイプコードとデータの一般的なコード化を参照する。

【００２２】実装貯蔵庫（インプリメンテーション・レポジトリ、implementation repository）50は、オブジェクト・サーバに関する情報を貯えるのに用いられる。特に実装貯蔵庫50は、サーバのプロセスをスタートさせるのに必要な情報を貯蔵している。例えば、実装貯蔵庫50はサーバ・プログラムのある場所、プログラムの引数、プログラム渡す環境変数などの情報を貯蔵している。

【００２３】簡便持続機能（単純パーシステンス、simple persistence）56は、IDLが定義するタイプとそれをIDLコンパイラに渡した結果を他のコードといっしょに用い、ディスク上に読み込んだり書き込んだりできるようにする。名前づけサービス（ネーミングサービス）52はORBオブジェクトを名前づけするのに用いられる。クライアントはこのサービスを用いて必要なオブジェクトを名前で検索することができる。名前づけサービス52ははオブジェクトへのレファランスを返すが、代わってそれはそのオブジェクトに何か情報を渡すのに用いられる。インタフェース貯蔵庫48（IFR）は分散オブジェクトシステム内にあるすべてのオブジェクトに関してのインタフェースを把握している。

【００２４】本発明は計算機システムの内部にあるデータとともにいろいろなステップをとる。これらのステップは物理量を操作する物理操作を要求する。通常これらの物理量は電子的あるいは磁気的な信号の形態を取ることが多く、これらは記憶されたり、転送されたり、あるいは相互に結びつけられ、比較されるなどの操作を受ける。これらの信号はビット、値、要素、変数、文字、データ構造などを単位に扱うのが便利である。それは他での使い方と共通性を持たせるためである。しかしながら、これら、あるいは他の同様な用語は常に適当な物理量と関連づけらけていることを忘れてはならず、これらの用語は実際の量を言及するための便利なラベルである

に過ぎない。

【００２５】さらには、行なわれる操作は、しばしば同定、実行（running）、比較などの用語を用いて述べられる。本発明の中に述べられるこれらの操作はいずれも機械の行なう操作である。本発明の操作を実行するのに適した計算機は、一般のデジタル計算機かあるいはそれと同様な装置である。いずれの場合にせよ、計算機に対して操作を実行させる方法と計算そのものの方法とは、異なることを常に心に銘記すべきである。本発明は計算機に操作をさせる方法に関するものであり、そこでは電気的かあるいは他の物理信号を処理し、望む形の物理信号を生成することになる。

【００２６】本発明は、これらの操作を実行する上での装置（apparatus） 器具にも関係している。この器具は特に特別な目的のために作られている可能性があるし、またどのような目的にも対応できるように計算機内のプログラムによって選んだ機能が作動したりその使途を再構成できるようになっているかも知れない。この中に記述されたプロセスは特に計算機やあるいは他の器具を限定していない。特に、各種一般的なマシンがここでの仕様に基づくプログラムで使われることもあれば、本プロセスを専用に実行できる特別な器具を構築すればより便利となることもある。これらマシンに要求される構造は以下に記述される。

【００２７】この中に記述された計算機に実装されるべき方法は、計算機科学の技法に基づき計算機システム上でプログラムを実行する良く知られた技術と器具を用いて実装可能である。ここで使われた「計算機システム」という用語は、処理装置（中央演算装置; CPU）、あるいはそれに相当するものを持ち、一つあるいは複数のデータ保持装置からデータおよび命令を読みだし、組み合わせ、処理する機構を指す。ここでデータ保持装置とはRAM、ROM、CD-ROM、ハードディスク、あるいは他の同様な機器を指す。データ保持装置は直接処理装置にかかることもあれば、ネットワークを介して遠隔の処理装置にかかることもある。遠隔のデータ保持装置がネットワークを介して遠隔の処理装置にかかるとき、特定のワークステーション上で実行できるようプログラムを送ることができればありがたい。加えて、処理装置が（並列処理装置として）同じ機械の中の他の処理装置と、あるいは（分散処理装置として）ネットワークを介して遠隔の処理装置と組合わさって動作することもある。このような遠隔地のデータ保持装置と処理装置の組み合わせは計算機技術の中の技法として一般的になっていくであろう。

【００２８】ここで「コンピュータ・ネットワーク」ということばは、相互につながった計算機システムとそれを結ぶ通信チャネル全体を含むものとして定義される。通信チャネルは、電線（ねじれ二重線）、同軸ケーブル、光ファイバー、衛星リンク、デジタル超音波など伝達メディアを指す。計算機システムは大規模に広域（数十、数百、数千マイル;WAN）に分散しているか、あるいは局地的な（数フィートから数百フィート;LAN）ネットワークに分散していることもある。さらにまた、いろいろなLANやWANが相互につながって計算機システムの集合体をなしていることもある。このひとつの例は「インターネット」と呼ばれるものである。

【００２９】本発明の実施例では、図２の100のところで図示したように、分散オブジェクトは相互に結合したネットワークの中の計算機の中にあってもよいし、一台の中にあってもよい。図のように、ネットワーク100は計算機102を含み、その計算機はさらにネットワーク104につながっている。ネットワーク104は、さらにサーバ、ルータ、あるいはそれに類した計算機106につながり、さらに計算機108、110、 112につながっている。データと実行命令はすべてこのネットワークを辿っていくことができる。コンピュータ・ネットワークの設計、構築、実装は一般的な技法となりつつある。

【００３０】計算機102、106、108、110、112は、図３の200のところで図式的に表現されている。計算機システム200は、プロセッサ（中央演算装置; CPU）202をある数含んでおり、それらは主記憶装置206（ランダム・アクセス・メモリ; RAM）、主記憶装置204（読みだし専用メモリ; RAM）を含んでいる。この技術ではよく知られているように、204はデータや命令をCPUに向かって単方向に送るのに対して、206は双方向に送ることができる。これら主記憶装置両方とも上に述べたようなメディアで計算機から読みだしができるようになっている。大容量記憶装置208は、典型的にはハードディスクのような二次記憶装置でプログラムやデータを貯えておくことができるが、主記憶装置に比べて遅い。大容量記憶装置208の中の情報は、場合によっては主記憶装置206の一部として仮想記憶となって使うことも通常行なわれる。CD-ROM 209のような特別な大容量記憶装置はCPUに一方向にデータを送る。

【００３１】CPU 202はインタフェース210を含む。インタフェースはビデオ・モニタ、トラック・ボール、マウス、キーボード、マイクロフォン、タッチパネル付き画面、カード読み取り機、磁気・紙テープ読み取り機、タブレット、尖筆(stylus)、声あるいは手書き文字認識装置、その他のよく知られた入出力が複数からなる。他の計算機がインタフェースになることもある。最後に、CPU 202は計算機や遠隔通信のネットワークと組合わさって用いられる。このネットワーク接続は、図の212で一般的に示されている。このようなネットワーク接続において、上記示した方法により、CPUは処理の過程を通じてネットワークから情報を受けとり、あるいはまた情報をネットワークに向かって発信することがわかる。今まで述べた機器や道具は計算機のハードウェアあるいはソフトウェアの技術として良く知られたものである。

【００３２】2. 分散オブジェクトシステムにインスト

ールされるアプリケーションを構築するコード生成器
図４の400は、分散オブジェクトシステムにアプリケーションを作成しインストールするシステムの実施例を摸式的に示している。図のシステムは、コンポジション構築器402（視覚的なアプリケーション構築器と呼ばれるものと同じ）を含み、ユーザ 特にプログラマーが、上記図１で示したような分散オブジェクトシステム上へインストールされるアプリケーションを作成するのに用いられる。このコンポジション構築器はコンポーネント・サービス404（後述）とともにユーザ（あるいはプログラマー）が分散オブジェクトシステム上で利用可能なオブジェクトをアクセスする機能を提供する。コンポジション構築器402は、さらにコード生成器408と連係し（これはさらにプログラムのテンプレート貯蔵庫406とつながっている）、コンポジション構築器によって作成されたものを410で示すようなプログラム・ソースファイルに仕立てあげる。

【００３３】プログラム・ソースファイル410は、さらにオブジェクト開発器ODFのコンパイラ/リンカ414に送られる。オブジェクト・アクセス・ソフトウェア412と接合したODFコンパイラ/リンカは、今度はコンポーネント・サービス404と連係する。ODFコンパイラ/リンカ414はオブジェクト・ソフトウェア416とネットワーク・アプリケーション418の両方を生成し、今度は420で示したネットワーク・オブジェクト422にアクセスする。これらのネットワーク・オブジェクトは、図の点線矢印で示したように、コンポーネント・サービス404かあるいはODFコンパイラ/リンカと連係して使われるオブジェクト・アクセス・ソフトウェア412に用いられる。オブジェクト・アクセス・ソフトウェア412は、典型的には削除されたヘッダ・ファイルかあるいは削除されたアプリケーション・プログラム・インタフェース（API）である。APIはクライアントにとって利用可能であり、ファクトリ・サービスを含むこともある。

【００３４】図４のこれらの要素をここで詳述する。コンポジション構築器402は、アプリケーション開発者に対して分散オブジェクトを用いて視覚的にアプリケーション・プログラムを構築することを可能にするものである。現存する、あるいは以前に開発されたオブジェクトのカタログがあるので、開発者は新しいアプリケーションプログラムを開発する際、容易に遠隔にあるオブジェクトを同定し取ってくることができる。加えて、これらのオブジェクトは新しいオブジェクトを作る際、構築器の中で再利用されるため、コードの再利用を許す。構築器の中であるオブジェクトが利用のために選ばれたとき、それはある種のサービスを提供することもあるし、あるいは逆に利用のためのサービスを要求することもある。例として、オブジェクトは何か値をとる属性を持つかも知れないし、何か引数をとって結果を返す操作（メソッド）を持つかも知れない。オブジェクトにより要求されるサービスとは、オブジェクトの属性に対しては値、操作に対しては引数である。同様に、オブジェクトによって供給されるサービスとは値が既に決まった属性であるし、操作結果である。アプリケーションを構築するときには、オブジェクトによって供給されるサービスは、それを処理上必要としているオブジェクトに「接続され」、配送されることもある。例えば、あるオブジェクトの属性値は他のオブジェクトの操作の引数として用いられるかも知れない。与えられたオブジェクトにより、どのサービスが供給されどのサービスが必要となるかを決定できるようにすることは、オブジェクトの結合上大事なことである。

【００３５】本発明の実施例では、「コンポーネント（componnent）」、「パーツ（part）」、「プラグ（plug）」、「ソケット（socket）」、「接続（connection）」は次の意味を持つ。 まず、コンポーネントは、あるオブジェクトあるいはオブジェクトタイプ、それが提供するサービス、その実装されたもの、および分散オブジェクトシステムの名前づけサービスにおけるそれの名前を表す。コンポーネントはコンポーネント・カタログに登録されており、ユーザがアプリケーションを構築するにときにコンポーネントを調べ、選択できる。ユーザが構築中のアプリケーションにコンポーネントを加えようとするとき、ここに説明されている構築のメカニズムによって特定メソッドを用いてコンポーネントの中にある情報から「パーツ」を引き出す。パーツとは、コンポーネントによって表されるオブジェクトの将来の実行時のインスタンスを保持する場所である。これは、他のオブジェクトと接続し属性値を定義するために実行時のインスタンスとして取り扱うことになる。構築器によってアプリケーションが完成し実行されるとき、パーツとともにそのアプリケーションにおける他のオブジェクトとの連携によるアクションを表現するコードが生成される。こうして最終的にアプリケーションが実行されるとき、オブジェクトの実行時インスタンスが生成されパーツに代わって配置される。

【００３６】ある実施例ではパーツは「プラグ」と「ソケット」を持つ。それぞれは視覚的にプラグ・アイコンとソケット・アイコンで表示される。上に述べたように、オブジェクトはその属性に対して値を要求し、操作に対して引数を要求する。「プラグ」とはこのようにオブジェクトにとって必要なサービスを指すのに用いられる。分散オブジェクト・システムの技法に熟練した人であれば、他の種類のサービスもプラグとして表現することができる。例えば、プラグはパーツの属性か、あるいは値を埋める必要のあるパーツの引数を表す。さらに、プラグはコンポーネントのファクトリ・メソッドの引数を表す場合もある。これらの引数は、コンポーネントのファクトリ・メソッドのによってパーツの実行時インスタンスを作るのに用いられる。

【００３７】同様にオブジェクトによって提供されるどのサービスも「ソケット」と呼ばれる。 例として、ソケットはパーツの既に値を持った属性を表す場合もあるし、ある値を返す操作を表す場合もある。このように、ソケットはプラグが必要としているサービスを表現している。プラグが必要している値を埋めるには多くの異なった型の値ががあり、かつソケットの方も異なった型があることが考えられる。例えば、値としてオブジェクト型を用いることがある。このようにプラグとソケットは相補的な性質を持ち、サービスを要求するオブジェクトとサービスを供給するオブジェクトの間の通信をを可能にすることがわかる。実施例ではこのような通信はプラグを第一のパーツ、ソケットを第二のパーツとした「接続」と表現される。

【００３８】上に述べたように、コンポジション構築器（composition builder）402は、分散オブジェクトのアプリケーションを作るための開発装置として用いられる。このモデルは、「コンポジション」と呼ばれ、コンポジション構築器のコンポジション・ワークシートによって形成される。このコンポジションは、分散オブジェクトのアプリケーションの図式表現とも考えられる。このコンポジションはパーツ、ソケット、プラグ、接続などの要素を含み、加えてインタフェースやモジュールの名前などのトップレベル情報も含む。このような要素すべては、シンボル表にロードされるときに値を決定する。このことは以下に述べる。

【００３９】典型的には、コンポーネント・サービス404を介して他の前からあるオブジェクトを用いて構築器（Builder）の中でオブジェクトを実装できる。より詳しく言えば、コンポーネント・サービスは、コンポジション・ワークシートの中でシンボル的に組み合わせることのできるコンポーネントを提供している。このコンポジションは実際のオブジェクトを含んでいるわけではなく、オブジェクトの型へのレファランスを持っている。コンポーネント・サービスは構築器に対してコンポーネントの記述を行なうインタフェースとそのための道具のセットである。

【００４０】構築器の中でコンポジションが完成したとき、多くの場合、コードに変換されコンパイルの対象になるようないろいろな「断片（piece）」ができ、それらはコード生成器408に送られる。コンポジションは、ユーザが使いたいと思っているパーツ、プラグ、ソケット、コンポーネントからなる拡張グラフを含んでいる。また、コンポジション・ワークシートの中で作られるオブジェクトの細かいインタフェース仕様の定義と、ときには作成中のオブジェクトで開発者が定義したメソッドを実装するためのソースコードが含まれることもある。これらの「破片」は、コンポジション構築器の中から「build」コマンドで実行され、コード生成器に送られる生の材料である。

【００４１】コード生成器は、コンポジション・サービスを介してコンポジションにアクセスするのが望ましい。コンポジション・サービスはある特定のコンポジションの中身を記憶するために使われるインタフェースのセットである。コンポジションは構築器が編集するコンポジションの下にある主「文書」である。コード生成器のAPIはネットワーク・オブジェクトであるから、「build」コマンドによって、コード生成器はコンポジションを完成するために、オブジェクトへのレファランスを渡される。そのときはいっしょにプログラム・テンプレートの貯蔵庫から適当なテンプレートが渡される。コード生成器がコンポジション・サービスからの情報を要求するとき、コンポジション・サービスのオブジェクトへのレファランスを通じて適当な呼びかけをすることにより、この情報にアクセスする。

【００４２】開発者が初期の段階で決定するのはクライアントとサーバのどちらを作るかである。すなわちコンポジション構築器は、クライアント・オブジェクトを作るのにもサーバ・オブジェクトを作るのにも使われるが、開発者は陽にどちらかの選択を行なう。一度この選択がなされると、プログラム・テンプレート貯蔵庫406は、クライアントなりサーバなりのテンプレートを提供することができる。プログラム・テンプレートは、クライアントなりサーバなりがどのような見えるかをレイアウトしたスケルトンのファイルである。しかし、これはコンポジション構築器のコンポジションとは結びつけられているわけでないから、まだ特定のアプリケーション・コードを持っていない。図５から図１２を通して、コード生成器がコンポジションをソースファイル410の形式のアプリケーション・プログラム変換していく過程を以下に詳しく述べる。

【００４３】プログラム・ソースファイルは、ODF414に供給され、次の二種類の出力をする可能性がある。まず、ODFはネットワーク・アプリケーション、すなわち分散オブジェクト・システムの中で走る「フロント-エンド」ソフトウェアを作成する場合がある。あるいはまた、ODFはオブジェクト・ソフトウェア416を作成する場合がある。オブジェクト・ソフトウェア416はオブジェクトにアクセスするために走るソフトウェアであり、必ずしもアプリケーションではない。ある実施例においては、オブジェクトのインタフェースはIDLで定義される。IDLはプログラミング言語に依存しない。このIDFは、ODF414に供給され、オブジェクト・ソフトウェア416を生成する。これはプログラミング言語に依存しないソフトウェアであり、IDLで定義されたオブジェクトにある属性や操作へアクセスすることができるようになる。すなわち、オブジェクト・ソフトウェア416は、IDLで定義されたオブジェクトにアクセスする分散オブジェクト・アプリケーションによって使われるソフトウェアのライブラリのように見える。

【００４４】図５の500のところは、本発明の実施例として広汎に分散オブジェクト・コードが生成されるようすを表している。コンポジションの構築に続いて、完成したコンポジションはコード生成器に渡されている。コード生成器はコンポーネントとコンポジション・サービスの「顔のない（faceless）」クライアントである。その仕事はコンポジションとコンポーネント・データからコンパイルされることになるコードを生成することである。これから創り出されるそれぞれのプログラム・ソースファイルにとっては、それに相当するテンプレートがあるのが望ましい。このテンプレートは、コード生成器がその中のメタシンボルを実際のコードに置換した結果である。コード生成器は、後に詳述するように、メタシンボルを置換するのを助けるのにシンボル表を用いる。言い替えれば、コード生成器はプログラム・テンプレート・ファイルとシンボル表を用い、これらプログラム・ソースファイルを作るものである。本発明の実施例としては、プログラム・ソースファイルは、C++のソースファイルであり、コンポジションと名付けられた分散オブジェクト・アプリケーションに対しては、これらのソース・ファイルは「composition.cc」、「compositon.hh」、「main.cc」、「Imakefile」を含む。

【００４５】本発明のコード生成器（コードジェネレータ）は、ステップ502で始まり、コンポジションができた知らせを受ける。典型的な例では、これは開発者がコンポジションを完成して「build」コマンドを発行したときである。ある実施例においてはコンポジションは視覚的なアプリケーション構築器と連結して作られる。このアプリケーション構築器は、分散オブジェクトのコンポーネントのカタログといっしょに用いられる。

【００４６】次にステップ504では、コード生成器はコンポジションをシンボル表の中にロードする。構築器の中でコンポジションから分散オブジェクト・アプリケーションを作るに際し、シンボル表がどのように使われるかは以下に詳述する。基本的には、ロードするステップでは、コンポジションから識別子（キーあるいは変数と呼ばれる）を取り出し、それらを対応する値にマップする。それぞれの識別子の値はシンボル表を介して記憶され、それぞれの識別子-値のペアはシンボル表のエントリと呼ばれる。それぞれの値はシンボル表、整数、文字列あるいは文字列ベクタである。これらの識別子は、シンボル表の値にマップされ、プログラム・テンプレートにとってシンボル表がアクセスできるようになり、ソース・プログラム・ファイルを作る上でそれ自身の識別子は適当な値に置換される。言い替えればこのコンポジションをロードするステップは、プログラムテンプレートの中で使われる識別子と値をシンボル表に「供給する（seed）」ことである。

【００４７】次にステップ506では、作られるべきファイル名のリストがそれぞれ対応するテンプレートを伴ってコンポジションから取り出される。このときそれぞれのファイル名に対応して一つのテンプレートがあることが望ましい。ファイル名のリストはコンポジションの中で暗に作られている。すなわち、開発者は最初クライアントを作るかサーバを作るか選択するときにどちらのテンプレートセットを使うか示しているのである。テンプレートは、希望のファイルを作るためにシンボル表の中の値と結びつけられる。コンポジションをシンボル表の上にロードし、ファイル名と対応するするテンプレートを取り出すと、次はステップ508でこれらりテンプレートとシンボル表からのシンボルの値を使ってプログラム・ソースファイルが生成される。ひとつのソースファイルはひとつのテンプレートから作られることが望ましい。このステップは図６とともに、以下により詳しく述べる。最後に、ソースファイルができあがってしまうと、ステップ510で示すように結果のファイルはODFに送られることがある。ODFは、ファイルを実行可能な形式にコンパイルしリンクする。

【００４８】図６の600の箇所は、本発明の実施例として606で示されるソースファイルを作るために604のシンボル表と602のプログラム・テンプレートを結び付けるようすを示している。一般に、テンプレートは結果となるソースファイルへと「マップ」される。テンプレートのある部分（ジェネリックなヘッダ部分など）は文字通りソースファイルにコピーされ、識別子などの他の部分はシンボル表により実際の値に置き換わる。この置き換えは識別子がシンボル表を検索したときに起こり、対応する値に変わる。（これはマクロ置換と呼ばれる） 例えば、サーバ・オブジェクトが作られるときには、そのインタフェースが定義される必要がある。このサーバのアウトラインのスケルトンは、テンプレート602で与えられる。ある実施例では、テンプレート602は、手続き呼び出しやメソッド・ルーチンに使われるジェネリック・ヘッダを含んでいる。602に示されているように、ひとつのヘッダは "class@(interface.module)::@(interface.name)" という形式の行である。この"@"シンボルは、それに続く識別子がシンボル表の中の値に置き換わることを示している。この例では、テンプレートの"class@(interface.module)::@(interface.name)"が、シンボル表の値に結びつけられる。シンボル表604は、識別子から値を与えるエントリを持っている。この例では、"interface.module"という識別子が"foo"という値を持ち、"interface.name"という識別子が"bar"という値を持っている。シンボル表の中のエントリは、テンプレートと結びつけられてソースファイルを作り、"class foo::bar"というコードを持つことになる。

【００４９】より洗練された方法で置換ができればなお良く、例で示したような特別な形式や構文でなくてもこれができればありがたい。例として、次に、どのようにこの置換が行なわれるのかをさらに説明する。上に述べ

たように、コード生成器はプログラム・ソースファイルとシンボル表を作る。テンプレート・ファイルは、"@"で始まる特別な形式を含んでいて、それがシンボル表からの値に置き換わる。テンプレート・ファイルの他の情報は、出力となるファイルにそのまま文字通りコピーされる。例えば、テンプレート・ファイルが、

My @(animal) has @(insect).

と言う行を含んでいたとすると、シンボル表"st"は次のように初期化される。

【００５０】

```
st.setValue("animal", "dog");
st.setValue("insect", "fleas");
```

コード生成器は、

My dog has fleas.

を生成する。

【００５１】"@(identifier)"という特殊な形式はシンボル表の中の識別子に対する値に置換される。識別子は任意の文字列であってよい。識別子はパス（path）を表現し、パス・コンポーネントはピリオドで区切られている。それぞれのパス・コンポーネントはシンボル表から分別して取り出され、最後のコンポーネント以外の値は全部シンボル表である。例えば 、@(foo.bar) の値は、@(foo)の値であるシンボル表の中の"@(bar)"の値を検索することによって取り出される。このようにして、「シンボル表」はシンボル表の木構造になっており、ピリオド"."で区切られたパスによってインデックスされている。

【００５２】次の例では、テンプレートファイルは次のように書かれてあるものとする。

【００５３】

```
The big @(story .homeowner) wanted to eat
@(story.protagonist).
```

またシンボル表 "st" は次のように初期化されているものとする。

【００５４】

```
CG::SymbolTable *story = new CG:: SymbolTable;
story.setValue ("homeowner", "brown bear");
story.setValue ("protagonist", "Goldilocks");
st.setValue ("story", story);
```

このとき、 コード生成器は次のコード

The big brown bear wanted to eat Goldilocks.

を生成する。

【００５５】シンボル表は、隠れていたシンボル表を自動的に生成できるのでより簡単な方法で初期化できる。

【００５６】

```
st.setValue ("story.homeowner", "brown bear");
st.setValue ("story.protagonist","Goldilocks");
```

シンボル表のエントリの値は再びシンボル表であるか、あるいは整数、文字列、文字列ベクタである。整数値を持つシンボル表のエントリは、@if/@else表現も用いることができる。例えば、ゼロでない値はtrueであるとき、

@if (flag) Hi Ho @else Hi No

は、flag の値が次のとき、

```
st.set.Value("flag",0);
```

"Hi No"となり、そうでないとき"Hi Ho"となる。コード生成器は、"@elseif"という表現も持ち、通常どおりに解釈される。

【００５７】ベクタの値を持つシンボル表は、@foreach という形式

The letters are: @foreach (letter all-letters) @(letter) @end.

を持つ。

【００５８】もし、シンボル表が次のように初期化されている場合、

```
CG::StringVector v;
v.insert("letterA");
v.insert("letterB");
v.insert("letterC");
st.setValue("all-letters", v);
st.setValue("letterA", "A");
st.setValue("letterB", "B");
st.setValue("letterC", "C");
```

コード生成器は次のようなコード

The letters are: A B C.

を生成する。

【００５９】@foreach 節のボディ（body）は文字列ベクタの中のそれぞれの項につき一回翻訳される。 ベクタのそれぞれの要素の（文字列）の値は@foreach ループの変数に束縛され、ループのボディの中で束縛子として用いられる。ループ変数が識別子に出会ったとき、それはその値に変換され、結果となる識別子はシンボル表の中から取り出される。例えば、ループの最初の一回めでletterの値はletterAになり、@(letter) は@(letter A)となり、 結果 A という値になる。

【００６０】@foreach節のボディの中では、文字列ベクタ: var:index, var:is-first,var:is-last のそれぞれの要素に対してさらに三つの変数が束縛される。これらの値はベクタの要素（いまループ変数 "var" に束縛される）に対する整数のインデックスであり、整数フラッグが "1" のとき最初かあるいは最後のベクタの要素が束縛されている。同じ ABC のシンボル表が与えられているとして、

```
@foreach (leter all-letters)
int @(letter) = @(letter:index) @if(letter:is-last);@else,@end @end
```

コード生成器は以下のコード

```
A=1, A=2, C=3;
```

を生成する。

【００６１】C の識別子を構成するときには、ひとつか

あるいはそれ以上の @if/@else/@end か @foreach@end 表現を用いると便利なことがある。この場合、コード生成器は、@{else}と@{end}を @else と @end のように扱う。そこで次のように書いた場合、

FOO@if(NEVER)HOOO@{else}BAR@{end}BAZ

"never" が "0" に束縛されたとすると、コード生成器は期待どおり"FOOBARBAZ"を生成する。

【００６２】コード生成器は一般目的のコードの生成器であるが、一応 C++/Objective Cのソースとヘッダ、Imakefileを生成することを意図したものである。また、識別子のコンポーネントは、次のように二重に間接的である。

【００６３】

@foreach (var list) list element @(var) is:

@(foo.var.value)

ここで、@(var)は新しい識別子のパスを作るために@(foo.bar.value)のように一度検索され、完全なパスの値が取り出される。

【００６４】一般にテンプレート・ファイルは、分散オブジェクトのアプリケーションを規範に基づき表現する。いろいろな種類の分散オブジェクト・システムや分散オブジェクト・アプリケーションになる可能性のあるテンプレート・ファイルの技術において、熟練している人たちに有意義であろう。テンプレート・ファイルの特別な例については、上で述べたとおりである。

【００６５】図に戻って、図７の700は図５の504で示したシンボル表の構築の実施例を表している。ステップ702で、探索パスがコンパイル・ツールによってロードされており、後に使うことになる。これらの探索パスは、コード生成器が用いるファイルのファイル名かディレクトリ名のリストである。このステップの一部として、コンポーネント・サービスがいくつかのコンポーネントを束ねることがある。例えば、二つの異なるコンポーネントが同じヘッダ・ファイルを使い、同じ探索パスの情報を記憶している場合がある ステップ704では、シンボル木の中に置かれるトップレベルのシンボルがロードされる。これらのトップレベル・シンボルは、それぞれのどのコンポジションにも関連づけられる大域的な名前である。いろいろな種類のトップレベル・シンボルがシンボル表の木にロードされる可能性がある。 例えば、これらのトップレベル・シンボルは、インタフェースの名前、インタフェース・モジュール、すべてのパーツのリストである可能があるし、あるいはもっと一般に、コンポジションのどのパーツ、プラグ、ソケットにも関連しない可能性もある。

【００６６】次にステップ706では、コンポジションに関係あるファイルのファイル記述子がロードされ、それらのうち適当なテンプレートが決定する。このステップは、図８でより詳細に説明される。ステップ708では、コンポジションで定義されたコンポーネントがロードされている。このステップは、コンポーネント・サービスに、コンポジションの中でパーツによって言及されているすべてのコンポーネントのリストを要求することで完結する場合がある。このステップは、図９においてより詳しく説明される。一度コンポーネントがロードし終ると、ステップ710 でこれらコンポーネントに対応するパーツがロードされる。このステップは、図１０ で詳細に説明される。ステップ712ではコンポジションの中で定義されたプラグ、ソケット、ベース・インタフェースがロードされる。このステップは、図１１でより詳細に説明される。ステップ718では接続がロードされる。このステップは、図１２および図１３でより詳細に説明される。

【００６７】図８は、ステップ706を詳細に説明したものである。このステップはコンポジションに関係したいろいろなファイルのファイル記述子をロードし、どのファイルがテンプレートであるか決定する。ファイル記述子は、ファイルを二つの次元から型に分ける。最初に、ファイルは中に何を含むかで分類される。ファイルは宣言、定義、IDLコードなどを含んでいる。次に、ファイルは何がもとになっているかで分類ができる。最初のカテゴリーでは、ファイルはコード生成器によって作られたものである。これらのファイルはフラグのついたテンプレートを持っていて、そのファイルを生成するのに敵したテンプレートを指定してある。二番目のカテゴリーは、開発者によって提供されたファイルであり、C++ のようなメソッドの定義を実装したソースコードを含んでいる。三番目のカテゴリーは、図４のオブジェクト・アクセス・ソフトウェア412のようにクロス・レファランスだけのものである。これらのクロス・レファランスにより、コード生成器はオブジェクト・アクセス・ソフトウェアで利用できるファイルを見つける技術を提供する。802から始めて、ファイルはまず生成されたものかそうでないかの類別を行なう。ステップ804では、非生成ファイルはさらに中身（内容の型（content type））による類別を行なう。プログラム・テンプレートのいくつかは非生成ファイル名を型によって言及するので、この方法は有用である。例えば、Imakefileはこれを行なう。

【００６８】ステップ806では生成されたファイルのリストが貯えられる。この生成ファイルのリストは図５の506で必要とされたとき、アクセスされ利用される。

【００６９】図９の900は図７のステップ708を詳しく説明している。このステップでは、コンポジションの中で言及されたコンポーネントをシンボル表にロードする。このステップではコンポーネント・サービスによって実行され、コンポジションの中で使われるすべてのパーツ、プラグ、ソケットその他をスキャンして、これらによって使われるすべてのコンポーネントのリストを作成する。いくつかのパーツは、同じコンポーネントを使う

ことがあるので、リストを作成する際は重複のないように、ちょっとしたチェックプログラムを走らせる必要がある。ステップ904では、コンポーネントの名前は対応する型とともにシンボル表に加えられる。コンポーネント・ファクトリは複合的な型であるので、このデータはいくつかの要素を含むことがある。これらはファクトリのクリエータの型とクリエータの名前を含み、ファクトリの名前は名前づけサービスによって登録される。ファクトリはコンポーネントのインスタンスを動作時に作るために用いられる。この実行時コードはインスタンスを作るために適当なファクトリを探す。

【００７０】図１０の1000は、図７のステップ710を詳細に述べたものである。このステップは、コンポジションの中で使われたすべてのパーツをシンボル表にロードする。1002から始めて、パーツのリストが決定する。このステップは、コンポジション・サービスが必要なすべてのパーツのリストを要求することで実行される。次にステップ1004では、パーツの名前が、パーツのコンポーネントへのクロス・レファランスを伴ってシンボル表に貯えられる。図９で各々のコンポーネントに対するシンボル表のエントリができるので、これらのシンボル表の名前は、それぞれのパーツのコンポーネントをクロス・レファランスするために貯えられる。ステップ1006では、それぞれパーツのクリエータ関数の引数が記憶される。それぞれのコンポーネントはファクトリを持っていて、それは今度はクリエータ関数になる。より正確に言えばステップ1006は、それぞれのパーツのコンポーネントのファクトリのクリエータ関数の引数を持っていることになる。このステップではシンボル表のエントリが作られて、この値はシンボル表で、そのファクトリのクリエータ関数によって必要とされるすべての引数へとマップされる。この引数の集合はファクトリ関数を呼び出すコードを生成するとき必要となる値の集合である。ステップ1008ではシンボル表にパーツの初期化する値を貯える。パーツのそれぞれの属性はこのステップで初期化される。

【００７１】図１１は、図７のステップ712を例示している。このステップでは、コンポジションの中で使われるすべてのプラグ、ソケット、ベース・インタフェースが、シンボル表に貯えられる。最初ステップ1102でプラグのリスト、ソケットのリスト、ベース・インタフェースのリストが決定する。これらのリストは、コンポジション・サービスをに問い合わせることで決定できることもある。一度これらの要素が決定すれば、次にステップ1104でそれぞれのプラグの名前と型、それぞれのソケットの名前と型、それぞれのベース・インタフェースの名前と型がシンボル表に読み込まれる（stored）。プラグやソケットの型は、IDL型のオブジェクトで、ソケットから渡されたり、プラグから受けとったりする。それは接続を受けるオブジェクトの型である。ベース・インタフェースは、他のインタフェースから導かれるものではない。 言い替えれば、 ベース・インタフェースでないインタフェースは他から引き出されたインタフェースである。

【００７２】図１２と図１３は、本発明の実施例に従って図７のステップ718を説明している。このステップは、コンポジションから接続をシンボル表に読み込んでいる。接続は、コンポジション構築器でアプリケーションを作成ときに開発者によって形成されるものである。接続はあるパーツのソケットから別のパーツのプラグへとつながるものである。慣例では、ソケットはプラグが必要としているオブジェクトを使うためにソケットがオブジェクトを提供するものである。これは電気機器のプラグとソケットを模倣したものである。電気的なプラグが電気的なソケットに差し込まれたとき、ソケットはプラグに電力を供給する。同様に、実行時には、接続はオブジェクトへのレファランスをソケットからプラグに渡すように見えるものである。さらに、接続はコンポジション・ソケットからパーツへ、あるいはコンポジション・プラグからパーツへとつながるものである場合もある。ここで「コンポジション」ソケット／プラグは、コンポジションの中から外へアクセスできるようにするものである。言い替えれば、コンポジション・ソケット/プラグは、コンポジションの公的なIDLインタフェースである。

【００７３】ステップ1202から始まって、接続のリストはコンポジション・サービスから得られる。この接続関係が一度得られれば、次のステップ1204から終了（END）までが実行され、それぞれの接続はシンボル表にロードされる。図１１で前に示したようにコンポジションのすべてのプラグとソケットは、既にシンボル表にロードされている。さらに、ソケットはオブジェクトへのレファランスをプラグに渡すようになっているが、これらのレファランスが実際に渡されるのは実行時である。さらに、正しいオブジェクトのレファランスが取り出されるためには、実行時に一連の操作がソケットに対して行なわれなければならないし、プラグに正しいオブジェクトのレファランスをを渡すためにも、プラグに一連の操作を施す必要がある。ステップ1204では、現時点での接続のプラグのクロス・レファランスがシンボル表に記憶される。次にステップ1206では、このプラグに対する操作列もシンボル表に貯えられる。このように、プログラムは適当なタイミングでこの一連の操作列（sequence of operation）が呼び出されて生成される。

【００７４】操作列の例を述べる。ソケットは何も引数をとらずに値を返す操作であるとする。すなわち、操作"part.get_value"は、何も引数をとらず値だけ返す。この構築器はユーザにソケットに対する操作列を作成できるようになっている。この操作は前のソケット操作の値に依存して変わるかも知れない。例えば、次の操作:

```
Type1 var1 = part.get_value1();
Type2 var2 = var1.get_value2();
Type3 var3 = var2.get_value3();
```

は、最終の値が"var3"になる操作列を定義している。

【００７５】ステップ1208は、その時点での接続のソケットのクロス・レファランスをシンボル表に記憶している。次のステップ1210では、このソケットに対する操作列がシンボル表に記憶される。次に、1214、1218、1222、1226のステップに従って接続がソートされる。接続は、その終端がソケットであるかどうか（コンポジション・ソケットがパーツに接続されたとき）、プラグであるかどうか（コンポジション・プラグがパーツに接続されたとき）、単に二つのパーツが接続されただけであるかどうか（あるパーツのソケッがもう一方でもう一つのパーツのプラグに接続されているとき）、あるいは接続のプラグ終端がそのパーツのクリエータ関数への引数になるものであるかどうかによってソートされる。

【００７６】最初ステップ1214では、接続の終端がコンポジション・ソケットであるかどうかを決定する。もしこのステップ1214の答えが"YES"であれば、ステップ1216においてソケットから接続へのクロス・レファランスがシンボル表内に生成される。このようにして、シンボル表の中にソケット記述へレファランスが付加される。いずれの場合にも、 ステップ1218で接続のどちらかの終端がコンポジション・プラグであるかどうかが決定される。もし1218の答えが"YES"であれば、ステップ1220において、プラグから接続へのクロス・レファランスが生成され、シンボル表の中に記憶される。さらに、この答えがいずれの場合でも、ステップ1222に進み、二つのパーツの間で接続があるかどうかが決定される。この質問1222の答えが"YES"である場合、ステップ1224においてこの接続が、初期接続を並べてあったシンボル表に加えられる。この初期接続は、アプリケーションが開始時に実行されるものである。もし1222の答えが"NO"であるなら、制御は次のステップ1226に直接進む。ステップ1226では、接続のプラグ終端が、そのパーツのクリエータ関数から導き出されたのであるかどうかを決定する。このとき、それぞれのパーツはクリエータ関数を持つコンポーネントに対応する。もしもこのクリエータ関数が引数を持つなら、これらの引数はそのパーツのプラグを形成する。なぜなら、それら引数はオブジェクトの値によって埋められなければならないからである。もしここでステップ1226で問題になっている接続のプラグ終端が存在すれば、パーツのクリエータ関数の引数を埋める（あるいは引数から導き出される）ことになり、次の制御ステップ1228に進む。ステップ1228ではシンボル表でパーツのクリエータ関数の引数リストに接続が付加される。ステップ1228ののち、あるいはもしステップ1226の答えが"NO"であれば、このルーチンは終る。

【００７７】前述の発明について明確な理解を得てもらうために説明をしてきたが、後に添付したクレームの範囲内である変更や改変がなされる必要があるのは明らかであろう。例えば、コード生成器はどの適当な分散オブジェクト・システムに対してもコードを生成する場合がある。プログラム・テンプレートで用いられるためにある特定のシンタックスが示されてきたが、これはどのようなシンタックスでもよい。加えてここに示したシンボル表は、識別子を値に結びつけるある一つの可能な場合を示した。この識別子を値に結びつけるやり方にはまだいろいろなやり方がありうる。さらに、シンボル表にいろいろな要素をロードする際にはある特定の順序を提示したが、異なる適当な順序でよい。したがって、ここで述べた実施例は例証するためのものであり、制限ではない。したがって、本発明はここに詳細に与えた方法に限らず、特許請求の範囲の主張によって、同じいかような方法であっても定義できる。

【図面の簡単な説明】

【図１】図１は、本発明に対応する分散オブジェクト・システムの概略図である。

【図２】図２は、本発明に対応するコンピュータ・ネットワークの構成図である。

【図３】図３は、本発明に対応するコンピュータ・システムのブロック図である。

【図４】図４は、本発明に関して，分散オブジェクトシステムにおけるオブジェクト指向アプリケーションの構築を説明する構成図である。

【図５】図５は、本発明に関するネットワーク・アプリケーションのコードを自動生成する方法を示すフローチャートである。

【図６】図６は、本発明の実施例において，テンプレートとシンボル表情報を組合せてソースファイルを生成する様子を示す構成図である。

【図７】図７は、図５のステップ504のフローチャートであり、 本発明の実施例において、 ネットワーク・アプリケーションがシンボル表にロードされるようすを図式化したものである。

【図８】図８は、本発明の実施例に関して図７のステップ706を詳細化したもので、 ファイル識別子がシンボル表にロードされるのを示す流れ図である。

【図９】図９は、本発明の実施例に関して図７のステップ708を詳細化したもので、 コンポーネントがシンボル表にロードされるのを示す流れ図である。

【図１０】図１０は、本発明の実施例に関して図７のステップ710を詳細化したもので、パーツがシンボル表にロードされるのを示す流れ図である。

【図１１】図１１は、本発明の実施例に関して図７のステップ712を詳細化したもので、プラグ、 ソケット、 ベース・インタフェースがシンボル表にロードされるのを示す流れ図である。

【図１２】図１２は、本発明の実施例に関して図７のス

27

テップ718を詳細化したもので、接続情報がシンボル表にロードされるのを示す流れ図である。

【図１３】図１３は、本発明の実施例に関して図７のステップ718を詳細化したもので、接続情報がシンボル表にロードされるのを示す流れ図である。

【符号の説明】

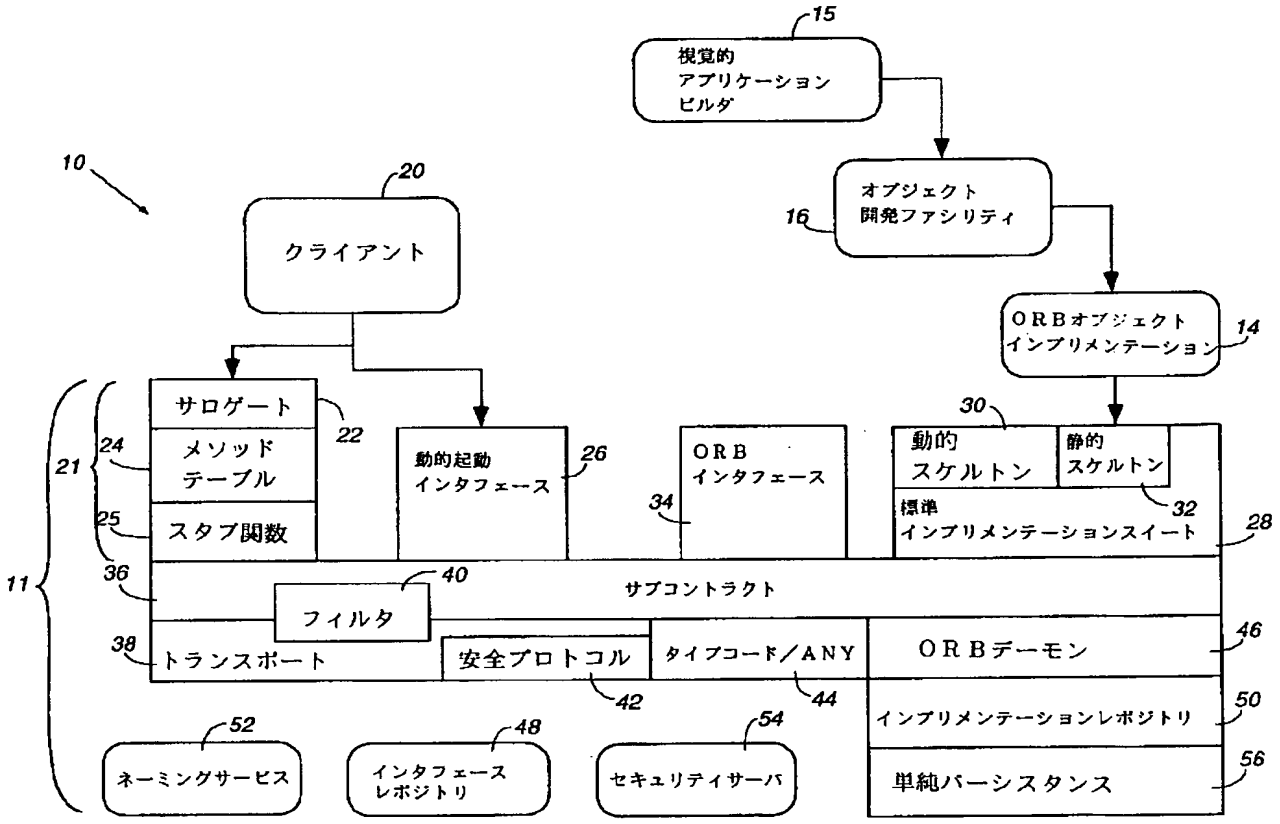
１０…分散オブジェクトシステム、１４…ＯＲＢオブジェクト・インプリメンテーション、１６…オブジェクト開発ファシリティ、２０…クライアント、２２…サロゲート

28

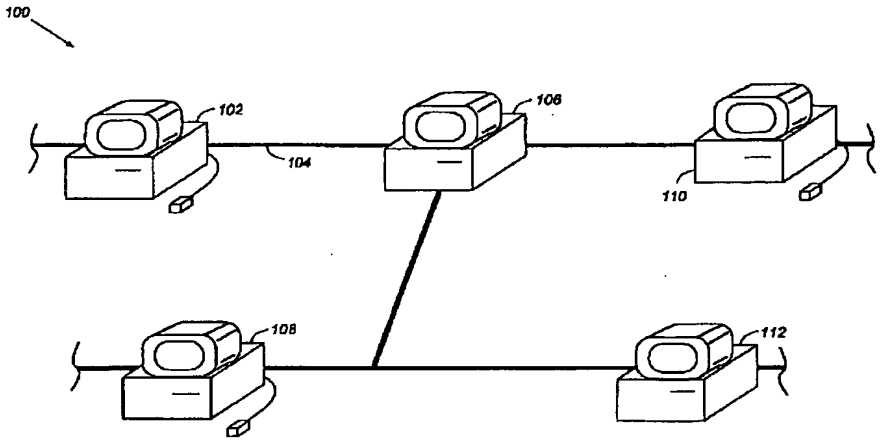
２４…メソッド表（メソッド・テーブル）、２５…スタブ関数、２６…動的起動インタフェイス、２８…標準インプリメンテーション・スイート

３０…動的スケルトン、３２…静的スケルトン、３４…ＯＲＢインタフェイス、３６…サブコントラクト、３８…トランスポート層、４２…安全プロトコル、４４…タイプコード／ＡＮＹ、４６…ＯＲＢデーモン、４８…インタフェース・レポジトリ、５０…インプリメンテーション・レポジトリ、５２…ネーミング・サービス、５４…セキュリティ・サーバ、５６…単純パージステンス

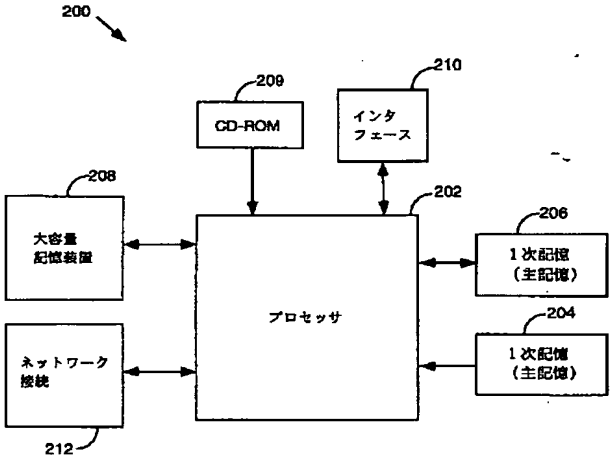
【図１】



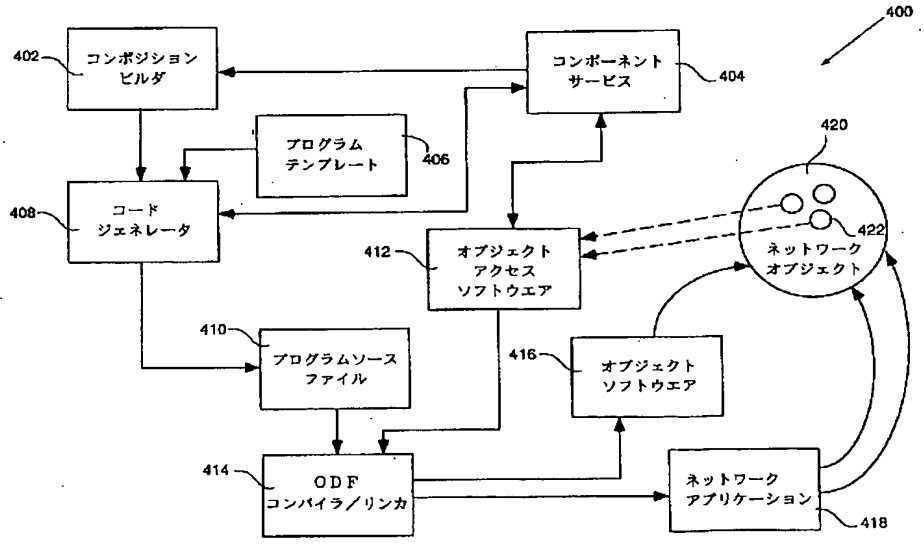
【図２】


100
102
104
106
108
110
112

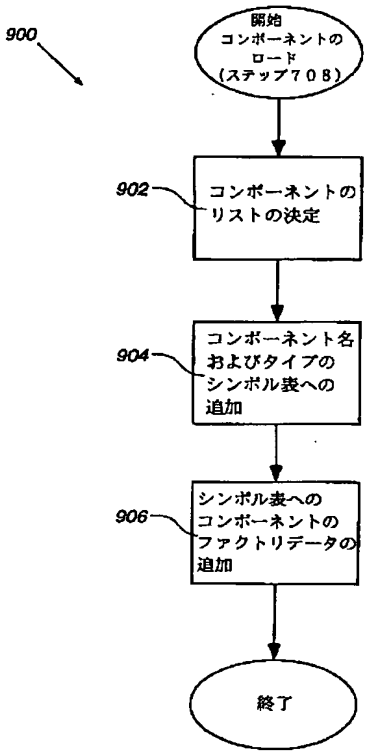
【図３】


200
209
CD-ROM
210
インタフェース
208
大容量記憶装置
202
プロセッサ
206
1次記憶（主記憶）
204
1次記憶（主記憶）
ネットワーク接続
212

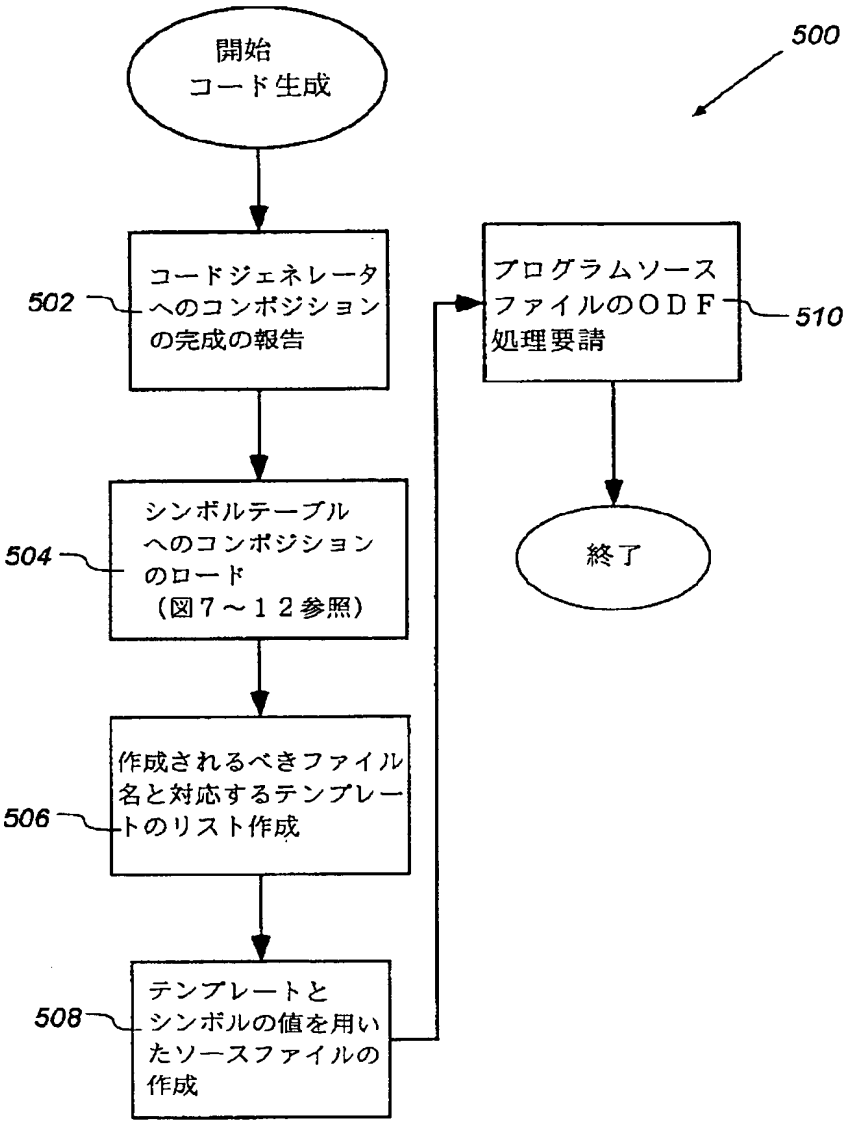
【図４】


400
402
コンポジションビルダ
コンポーネントサービス
404
プログラムテンプレート
406
420
408
コードジェネレータ
412
オブジェクトアクセスソフトウエア
422
ネットワークオブジェクト
410
プログラムソースファイル
416
オブジェクトソフトウエア
414
ODFコンパイラ／リンカ
ネットワークアプリケーション
418

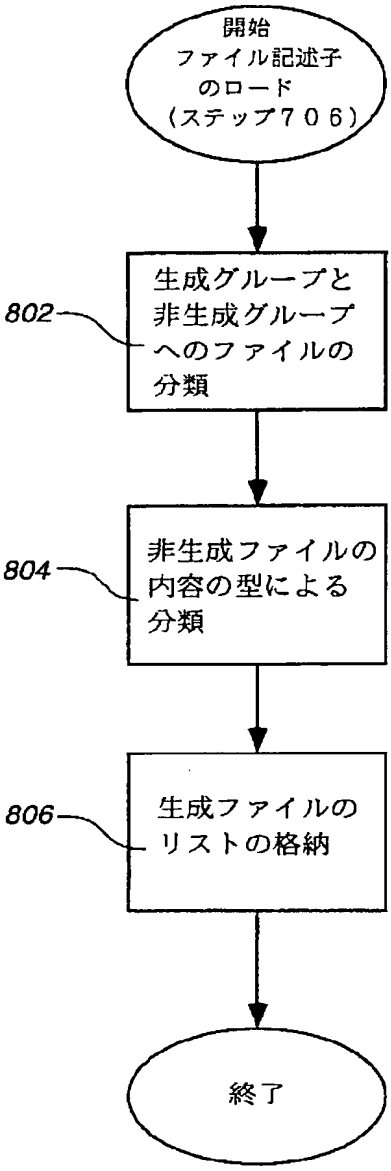
【図９】


900
開始
コンポーネントのロード（ステップ７０８）
902
コンポーネントのリストの決定
904
コンポーネント名およびタイプのシンボル表への追加
906
シンボル表へのコンポーネントのファクトリデータの追加
終了

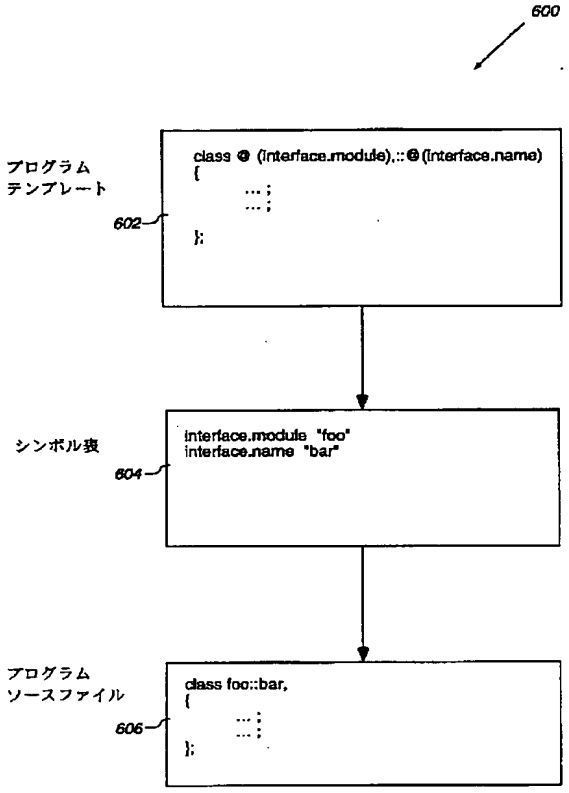
【図５】


開始
コード生成
500
502
コードジェネレータへのコンポジションの完成の報告
504
シンボルテーブルへのコンポジションのロード
（図７～１２参照）
506
作成されるべきファイル名と対応するテンプレートのリスト作成
508
テンプレートとシンボルの値を用いたソースファイルの作成
プログラムソースファイルのＯＤＦ処理要請
510
終了

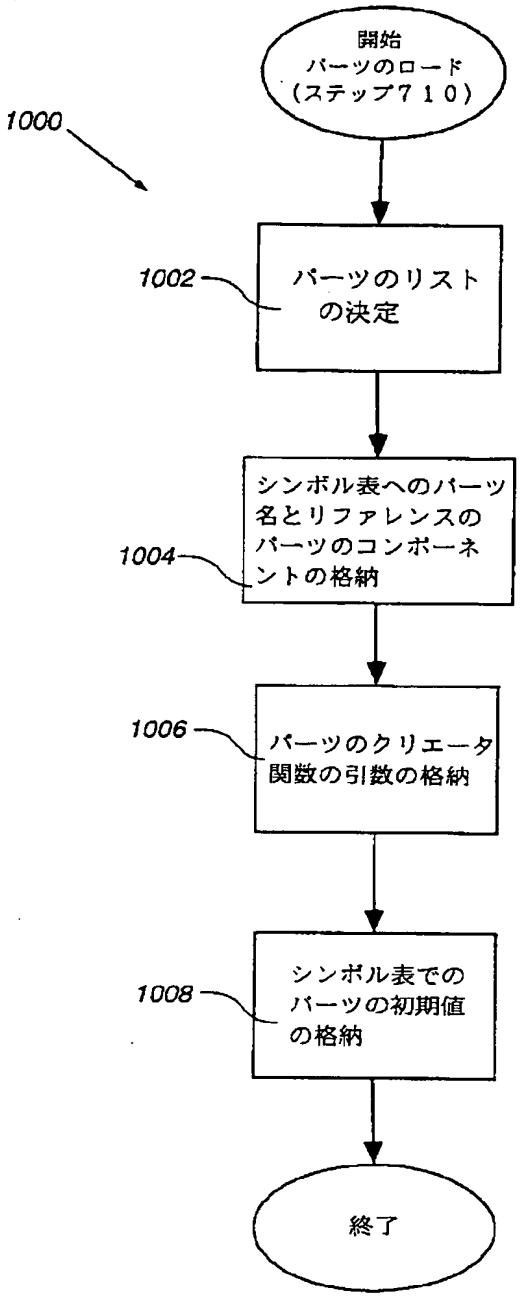
【図８】


開始
ファイル記述子のロード
（ステップ７０６）
802
生成グループと非生成グループへのファイルの分類
804
非生成ファイルの内容の型による分類
806
生成ファイルのリストの格納
終了

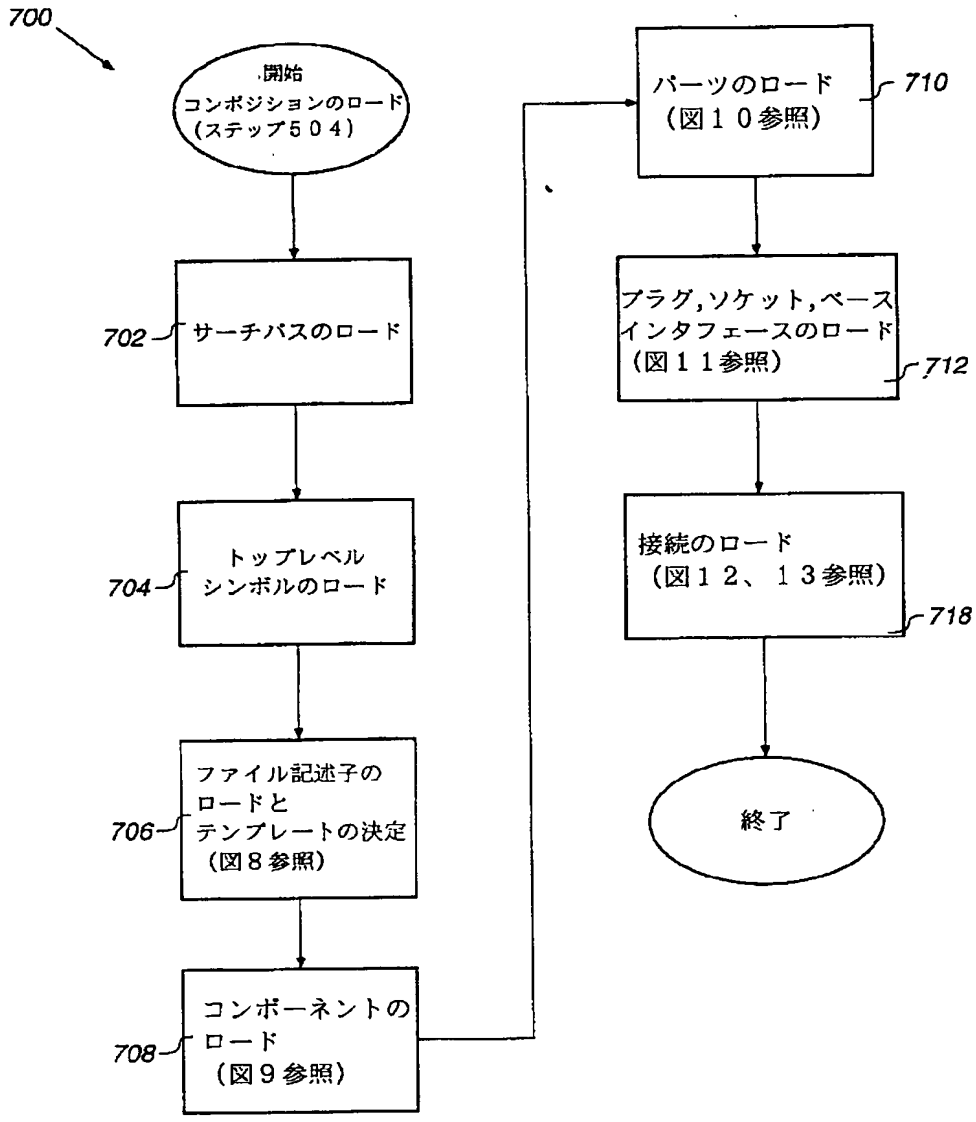
【図６】


600
プログラム
テンプレート
602
class ❷ (interface.module),::❸(interface.name)
{
};
シンボル表
604
interface.module "foo"
interface.name "bar"
プログラム
ソースファイル
606
class foo::bar,
{
};

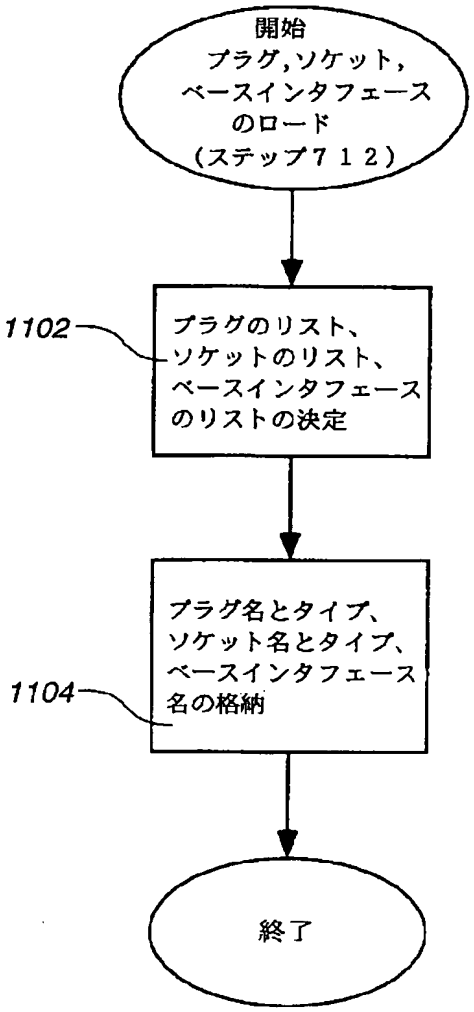
【図１０】


開始
パーツのロード
（ステップ７１０）
1000
1002
パーツのリスト
の決定
シンボル表へのパーツ
名とリファレンスの
パーツのコンポーネ
ントの格納
1004
1006
パーツのクリエータ
関数の引数の格納
1008
シンボル表での
パーツの初期値
の格納
終了

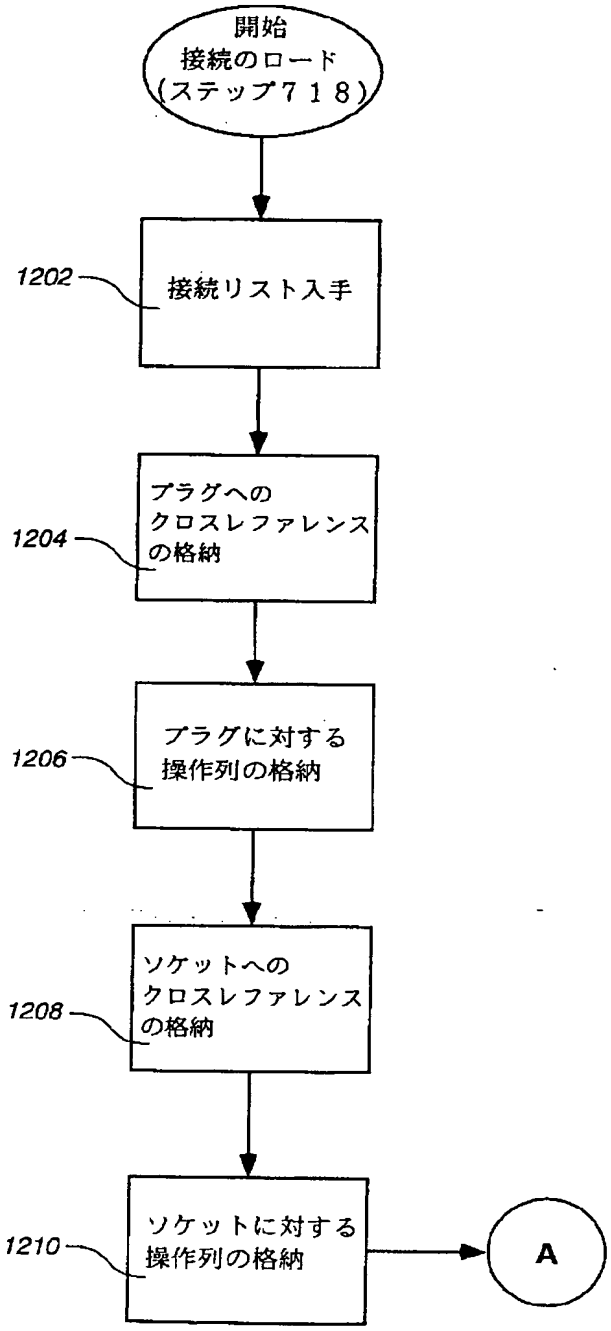
【図７】


700
開始
コンポジションのロード
（ステップ５０４）
702
サーチパスのロード
704
トップレベル
シンボルのロード
706
ファイル記述子の
ロードと
テンプレートの決定
（図８参照）
708
コンポーネントの
ロード
（図９参照）
710
パーツのロード
（図１０参照）
712
プラグ，ソケット，ベース
インタフェースのロード
（図１１参照）
718
接続のロード
（図１２、１３参照）
終了

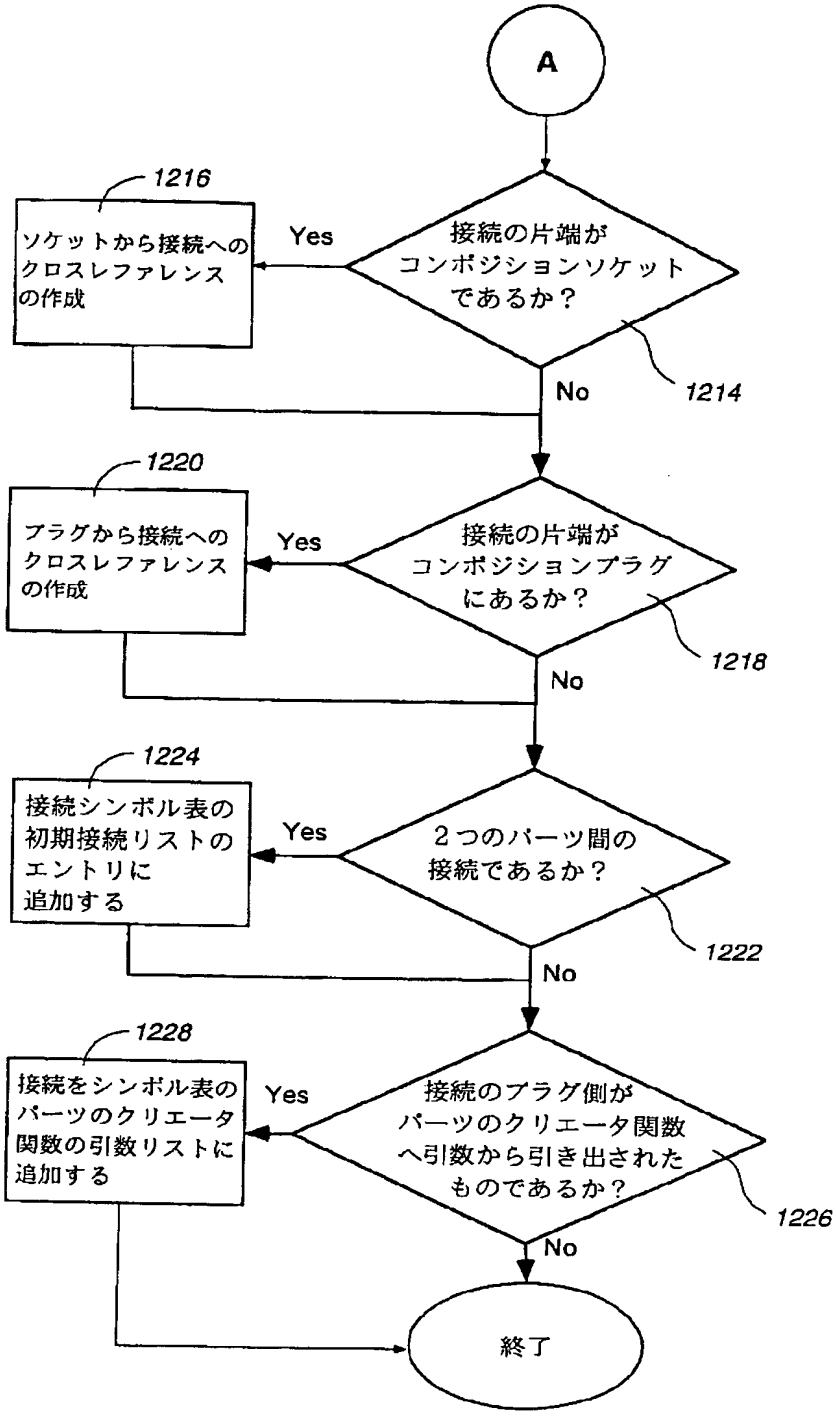
【図１１】


開始
プラグ,ソケット,
ベースインタフェース
のロード
(ステップ７１２)
1102
プラグのリスト、
ソケットのリスト、
ベースインタフェース
のリストの決定
プラグ名とタイプ、
ソケット名とタイプ、
ベースインタフェース
名の格納
1104
終了

【図１２】


開始
接続のロード
(ステップ７１８)
1202
接続リスト入手
プラグへの
クロスレファレンス
の格納
1204
プラグに対する
操作列の格納
1206
ソケットへの
クロスレファレンス
の格納
1208
ソケットに対する
操作列の格納
1210
A

【図１３】


A
1216
ソケットから接続へのクロスレファレンスの作成
Yes
接続の片端がコンポジションソケットであるか？
No
1214
1220
プラグから接続へのクロスレファレンスの作成
Yes
接続の片端がコンポジションプラグにあるか？
No
1218
1224
接続シンボル表の初期接続リストのエントリに追加する
Yes
２つのパーツ間の接続であるか？
No
1222
1228
接続をシンボル表のパーツのクリエータ関数の引数リストに追加する
Yes
接続のプラグ側がパーツのクリエータ関数へ引数から引き出されたものであるか？
No
1226
終了

フロントページの続き

(72)発明者　グレゴリー　ビー．　ニュイエンズ
アメリカ合衆国，　カリフォルニア州，
メンロ　パーク，　ローレル　アヴェニュー　403

(72)発明者　ハンズ　イー．　ムッラー
アメリカ合衆国，　カリフォルニア州，
サラトガ，　メローウッド　ドライヴ
12160